Chapter 14
Scrum Planning Principles

An old myth states that development with Scrum takes off with no planning. We just start the first sprint and figure out the details in flight. This isn't true. We do real planning in Scrum. In fact, we plan at multiple levels of detail and at many points in time. To some it may seem like Scrum deemphasizes planning because a majority of the planning occurs just in time instead of substantially up front. In my experience, however, teams often spend more time planning with Scrum than with traditional development; it just might feel a bit different.

In this chapter I expand upon several of the Scrum principles described in Chapter 3, with a focus on how they apply to planning. In doing so, I set the foundation for the discussion in Chapter 15 of the multiple levels at which Scrum planning takes place. In subsequent chapters I will explore in greater detail portfolio planning, product planning, release planning, and sprint planning.

Overview

Chapter 3 described key Scrum principles, a number of which are fundamental to how we approach planning when using Scrum. This chapter emphasizes the principles shown in Figure 14.1.

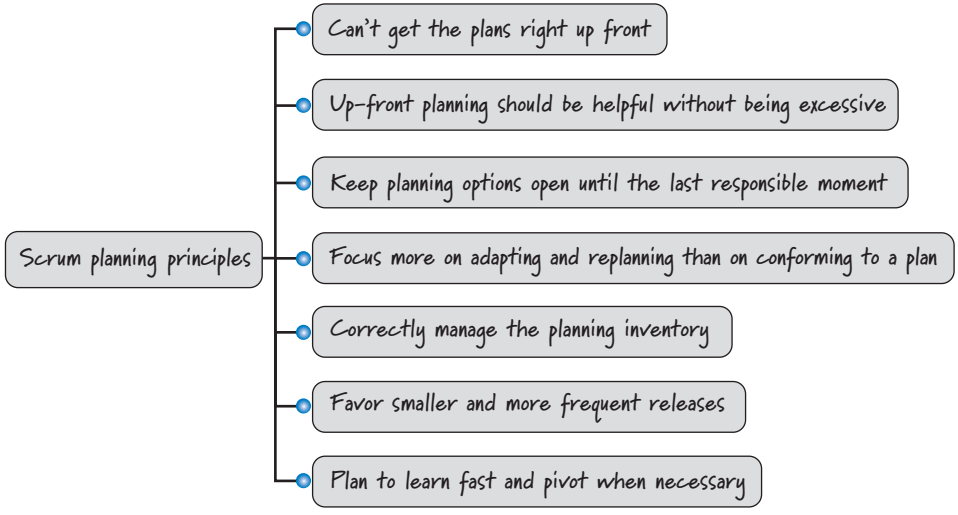

FIGURE 14.1 Scrum planning principles

Additional Scrum principles (such as work in short timescales, leverage cadence, and others) will be emphasized in subsequent planning chapters.

Don't Assume We Can Get the Plans Right Up Front

The traditional, predictive approach to planning is to create a detailed plan up front before development work begins. The goal is to get it right so that the rest of the work can proceed in an orderly fashion. Some people argue that without this plan we won't know where we are going and cannot coordinate the people and their activities, especially on larger development efforts with multiple teams. There is truth in that argument.

The Scrum approach to planning is true to its empirical roots of inspection and adaptation. When developing using Scrum, we don't believe we can get it right up front, so we don't try to produce *all* of the planning artifacts up front. We do, however, produce *some* of the planning artifacts early on in order to achieve a good balance between up-front and just-in-time planning.

Up-Front Planning Should Be Helpful without Being Excessive

Let's look at an example that illustrates the principle that *up-front planning should be helpful without being excessive*.

I live in Colorado where the skiing is world-class. Occasionally I do some recreational skiing, but I'm no expert. A friend of mine, John, is an extreme skier. Honestly, there are times I wish I were one as well, but I'm not that skilled or that crazy; John is. Once John was sharing pictures of his adventures on a particularly insane mountain. Out of curiosity I asked him a simple question: "When you are positioned at the top of the mountain preparing to start your run, do you plan your entire route down?"

After he finished chuckling, he remarked, "No, if I did that I would just get myself killed." He went on to say, "I pick a spot some distance down the mountain. My first goal is to ski to that spot. Maybe I plan the first two or three turns. Realistically, planning any further would be impossible and dangerous."

"Why?" I asked.

"The terrain isn't what it appears to be from the top because the lighting and other factors play tricks on you. Also, there are probably some trees down there somewhere, but from up top I can't see them—if I decide while standing at the top to turn right at some point, and actually follow through on it, I might fly right into those trees. Plus there's no predicting the 15-year-old on a snowboard who will fly over my head yelling, 'Watch out, dude!' You never know when you'll have to change course, or why."

After he finished his explanation, I remembered thinking, "Wow, that sounds like every interesting product development effort I've worked on." You never can predict with great certainty when you'll have to change course, or why. On the products where I was asked to create a detailed up-front plan, I did create one. But I can't recall a single time this approach worked out. After we finished, I can't recall ever looking back at the original plan and saying, "Got it perfect!" In a sense, trying to do too much up-front planning is like trying to plan every turn while standing on top of the mountain. Planning at this level of detail is wasteful; believing the plan is correct to the point of ignoring real-time data is downright dangerous.

Most of us have been involved in developing products where the level of detail associated with up-front predictive planning was just absurd. Does that mean we should do no up-front planning? No, that would be negligent and foolhardy. John certainly does some up-front planning—he studies major features of the terrain to give himself confidence before he starts his run. Equally foolhardy, however, is to plan to the point where it is difficult or costly to react to reality. Like John, we must find the proper balance between up-front prediction and just-in-time adaptation.

Keep Planning Options Open Until the Last Responsible Moment

To achieve a good balance between up-front and just-in-time planning, we are guided by the principle that we should keep important options open until the last responsible moment. This means we save the planning that is best performed in a just-in-time fashion for a time when we have much better information. Why make an early planning decision based on poor information? In addition to being very costly, premature decision making can also be dangerous, as John pointed out.

Focus More on Adapting and Replanning Than on Conforming to a Plan

One of the issues on many product development efforts is that too much emphasis is placed on the up-front plan and not enough on continuous planning. If we spend significant time up front developing a highly predictive plan, and we believe we got it right, there will be significant inertia to conform to the plan instead of updating it to respond to change. If we believe instead, as we do with Scrum, that you can't get the plans right up front and you can't eliminate change, we will value responding to change and replanning over following the up-front plan.

In the 1980s I spent time either helping develop large plans or working as a consultant to companies that had developed such plans. You know the type of plans I'm talking about—the ones yielding a large Gantt chart that we print (across multiple pages), tape together, and hang on the wall (see Figure 14.2).

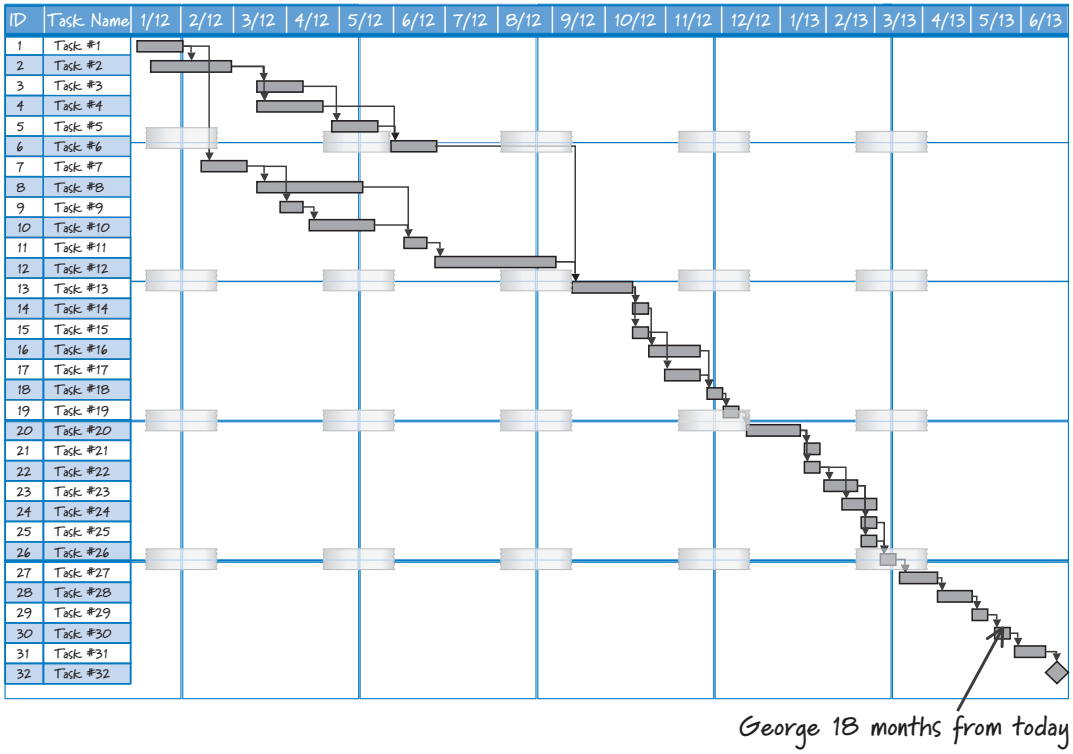

FIGURE 14.2 Big up-front Gantt chart

On several of these development efforts we spent up to six weeks developing highly predictive up-front plans. Once we produced these plans, they became the maps for the projects. Much as a legal system might assume "innocent until proven guilty," these plans were "assumed correct until proven wrong." A bit of wisdom often attributed to the Swiss Army, but more likely derived from the *SAS Survival Guide* (Wiseman 2010), seems appropriate here: "When lost in the woods, if the map doesn't agree with the terrain, in all cases believe the terrain." (See Figure 14.3.)

On any given product, our misguided faith in the map leads us to conclude that progress can and should be measured as conformance to or variation from the plan. When plan deviations occur, our desire for plan conformance blinds us to the fact that the map itself could be wrong. If the map becomes more important than the terrain, we have lost touch with the reality in which we must navigate.

When using Scrum, we view the up-front plan as helpful, but we believe that reading and adapting to the terrain are necessities. This is a reasonable belief when you consider that any up-front plan is put together when we have the least possible knowledge that we will ever have about our product. As such, up-front plans very accurately encode our early ignorance.

FIGURE 14.3 When the map and the terrain don't agree, believe the terrain.

In Scrum, we favor frequent replanning as we validate our assumptions. We use our validated learning to continuously produce better, more useful plans. We don't worry about our plans being wrong, because we know we will soon replace them with more accurate plans. Because we work in short sprints of a few weeks to no more than a month, even if we're wrong, we won't spend too long going down the wrong path before adjusting course.

Although most Scrum teams I have seen don't employ a Gantt chart, they do plan and do value having some form of longer-term planning. In fact, as I will discuss in Chapter 15, Scrum teams plan at multiple levels of detail. What we don't want is to get so wedded to our plans that we aren't willing to replan when things change or when we learn important information to which we must react.

Correctly Manage the Planning Inventory

In Chapter 3 I discussed a key Scrum principle of managing inventory (also referred to as work in process or WIP). When determining the proper balance between up-front and just-in-time planning, a key insight is to realize that creating a large inventory of predictive, not-yet-validated planning artifacts is potentially very wasteful.

Correctly managing our inventory of planning artifacts is the economically sensible thing to do.

Take the previous example of the large Gantt chart produced up front. As the development effort unfolds and we validate our assumptions by acquiring knowledge about what we're doing, we'll learn where our original plan was wrong. Unfortunately, by that time we're saddled with the waste of having to unwind and redo the future plans that have been invalidated by what we just learned.

This generates at least three forms of waste. First, there is the wasted effort to produce the parts of the plan that now have to be discarded. Second, there is the potentially significant waste of having to update the plan. And third, there is the wasted opportunity of not having invested our time in more valuable activities (like delivering high-value, working software) instead of doing up-front work that then requires future work to fix.

I always try to balance how much planning I do at a given time against the probability that what I'm doing will amount to waste if there is a change. For example, in the Gantt chart in Figure 14.2, I could have George's name next to a task that is going to start 18 months from today. What do you think the chances are that George will work on that specific task 18 months from today? Probably close to zero percent!

So, if there is such a high probability of the plan being wrong that far into the future, why plan that far out? Usually because we are trying to answer questions like "When will we be done?" or "How many people will we need for this development effort?" Unless we predict all of the work, how can we answer these questions with any certainty?

When a product will ship and which features we can get into a product by a predetermined date are valid questions that need to be addressed. However, we can't deceive ourselves into thinking we have the right answer just because we did long-term, low-certainty guessing. I will address these planning questions in the next several chapters.

Favor Smaller and More Frequent Releases

Scrum favors smaller, more frequent releases because they provide faster feedback and improve a product's return on investment (ROI). We can almost always improve the lifecycle profits of our product by leveraging incremental development and multiple releases of smaller marketable subsets of features.

Consider the economics of a single-release product, as shown in Figure 14.4 (from Denne and Cleland-Huang 2003). At the beginning of development we are spending money (starting the investment period) without any return. The release of the product occurs on the downward slope of the curve during the investment period. Eventually we achieve self-funding, when the product's revenue equals the

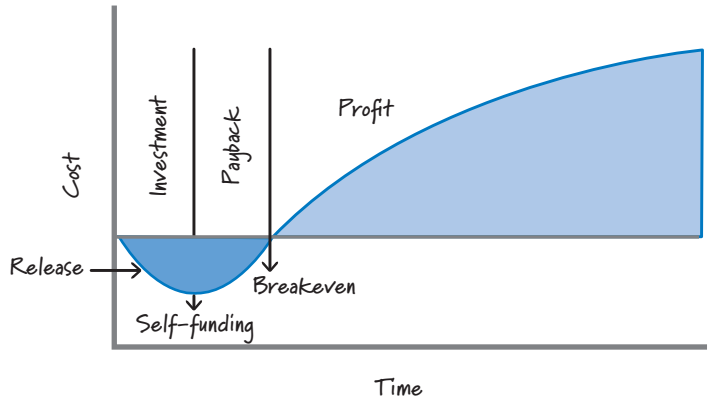

FIGURE 14.4 Single-release economics

cost of development. Once revenues exceed costs, we enter the payback period, where we start to recoup our investment. When total revenue equals total costs, we have achieved the breakeven point. From that point forward we are finally profitable!

To illustrate the benefits of smaller, more frequent releases, assume we release twice instead of once (see Figure 14.5). In this case we reach self-funding, breakeven, and profitability sooner, thus improving the overall ROI for the product.

As a specific example (adapted from Patton 2008), look at the ROI improvement of a model with the assumptions shown in Table 14.1.

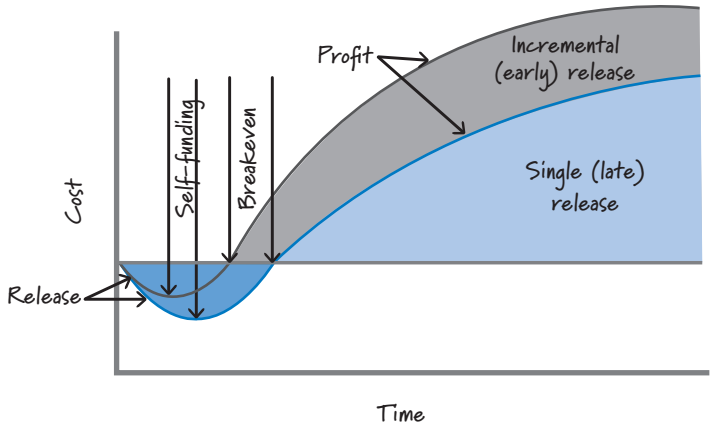

FIGURE 14.5 Multi-release economics

TABLE 14.1 ROI Model Assumptions

Variable	Value
Revenue (all features)	$300K/month
Revenue (1/2 features)	$200K/month
Revenue (1/3 features)	$150K/month
Delay from delivery to revenue	1 month
Development cost	$100K/month
Release cost	$100K per release

TABLE 14.2 ROI of Different Release Cycles

	Single Release (12 months)	Semiannual Releases	Quarterly Releases
Total cost	$1.3M	$1.4M	$1.6M
Total two-year return	$3.6M	$4.8M	$5.25M
Net two-year return	$2.3M	$3.4M	$3.65M
Cash investment	$1.3M	$0.7M	$0.45M
Internal rate of return (as a surrogate for ROI)	9.1%	15.7%	19.5%

The results, shown in Table 14.2, illustrate that, with a single release, after 12 months we have an ROI of 9.1%. If instead we do two semiannual releases, the ROI improves to 15.7%; with quarterly releases we achieve an ROI of 19.5%.

There are some limitations to this approach. First, for any product there is a minimum releasable or marketable set of features. Therefore, we can't just continue making the initial release smaller, because eventually it becomes so small as to not be marketable. Also, in some markets smaller and more frequent releases might not be an option. However, if your marketplace is open to receiving partial value sooner, delivering smaller and more frequent marketable releases is a very important principle to follow.

Plan to Learn Fast and Pivot When Necessary

There is no amount of up-front predicting or guessing that will replace doing something, learning fast, and then pivoting if necessary. By pivoting I mean changing directions while staying grounded in what we've learned. Ries defines pivoting to be

"a structured course correction designed to test a new fundamental hypothesis about the product, strategy, and engine of growth" (Ries 2011). Just like John the skier, we need to be prepared to pivot quickly when we learn that our current plan is no longer valid.

As I discussed in Chapter 3, our goal is to move through the learning loop quickly and economically. So we should structure our plans with learning as a key goal. By getting fast feedback, we can determine whether our plans are taking us in a viable direction. If not, we can pivot or redirect ourselves.

Closing

In this chapter I discussed and provided an overview of several Scrum planning principles. These principles enable us to plan in an economically sensible fashion by doing a helpful amount of up-front planning, balanced with more detailed, just-in-time planning as we learn more about what we are building and how to build it. In the next five chapters I will illustrate with examples how to leverage these principles at a deeper level in the context of the multiple levels of Scrum planning.